13

Model-based Reinforcement Learning with Model Error and Its Application

Yoshiyuki Tajima and Takehisa Onisawa
University of Tsukuba, Japan

1. Introduction

Many systems working with humans, e.g., humanoid robot, environmental intelligence, portable information assistant (Isozumi et al., 2003), have been studied in recent years. These systems need autonomous learning approaches for adaptation to various users' requests and environment changes. One of approaches to autonomous learning is Reinforcement Learning (RL) (S.Sutton et al., 1998). Study on shooting robot is an example of its early applications to practical problems (Asada et al., 1994). This study shows that RL can be used for autonomous systems. RL, however, cannot be applied easily to every practical problem since RL needs much time for learning. Therefore, other methods instead of RL are necessary for practical applications. One of the methods is Model-based RL, which has an inner model of environment and improves the learning efficiency by this model. That is, if an agent can get knowledge of a task and environment, then the agent uses it for learning. Model-based RL, however, has also some another problem when the inner model has some model error, because the agent learns some behavior by the inner model with the error. Although this problem is inevitable for Model-based RL, there are few studies on this problem.

This chapter gives careful consideration on the effects of the model error for Model-based RL and proposes a new approach, Model Error based Forward Planning Reinforcement Learning (ME-FPRL) (Tajima, et al., 2006) to solve the abovementioned problem. ME-FPRL controls learning based on errors of the inner model and can make the learning efficiency high. And ME-FPRL is applied to the pursuit of a target by a robot camera. The results of this application show that ME-FPRL learns more efficiently than usual RL and Model-based RL. Finally, conclusions and future work are shown.

2. Model-based reinforcement learning

This section introduces the framework of Model-based RL and describes one of Model-based RL algorithm as a preparation for ME-FRPL.

2.1 Framework of RL and model-based RL

Fig.1 shows the framework of RL. RL is based on Monte Carlo Methods and Dynamic Programming. An agent that learns with RL acquires some appropriate actions. In the RL method, it takes an action to environment. Environment changes by the action of the agent and gives reward to the agent according to the environment change. After the agent gets reward, the agent evaluates the action. Then, the agent observes the new state of environment and takes a new action to environment in order to get more reward. The agent learns the appropriate behavior by these interactions with environment and finally acquires the behavior giving maximum rewards.

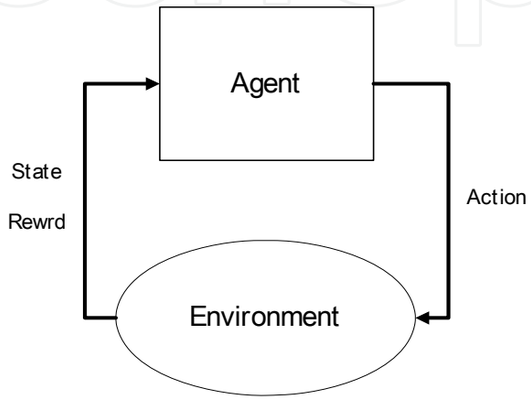


Fig.1. Framework of RL

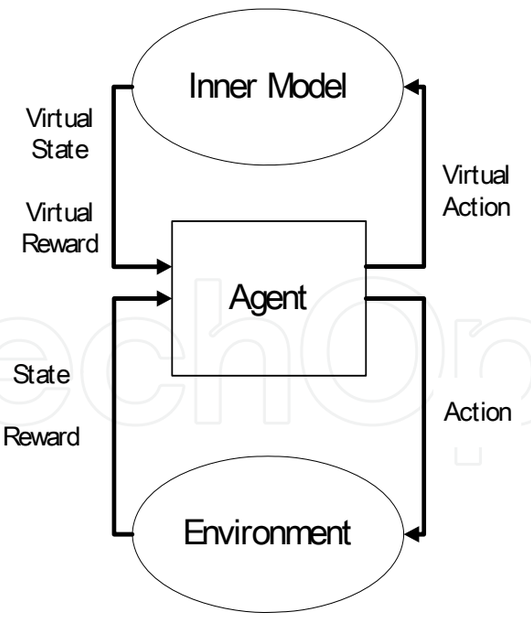


Fig. 2. Framework of Model-based RL

Fig.2 shows the framework of Model-based RL. An agent that uses the Model-based RL method has an inner model of environment and improves learning efficiency using the inner model. Model-based RL, however, has some problems about the error of the inner model.

The agent learns the behavior by both interactions with real environment and those with the inner model. The learning by the interaction with environment is called direct learning, and the learning by the interaction with the inner model is called indirect learning. If the agent's model is correct, that is, the same as environment, the agent can get a good performance by indirect learning because the model generates good experiences for the learning. On the other hand, if the inner model has some errors, or has some differences from real environment, the agent cannot learn the behavior correctly and the learning efficiency becomes low. It is found that Model-based RL has the problem on the error of the inner model.

2.2 Value function

In the RL algorithm, knowledge is expressed by a value function. The value function $V^{\pi}(s)$ is defined as Eq. (1), where $s \in S$ is the state of environment, $a \in A$ is the action taken by an agent, r is a reward given to an agent according to environment changes, t is the step of time, $\pi(s,a)$ is a policy and E_{π} is the expectation of a policy.

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\} \tag{1}$$

An action-value function $Q^{\pi}(s,a)$ is defined as Eq. (2). The action-value function is often used for expressing knowledge because it is easy to find the best action on the current state. The best action a^{*} is obtained by Eq. (3). When $Q^{\pi}(s,a)$ is optimized and the agent continues taking a^{*} each state, the agent can reach goal efficiently.

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\} \tag{2}$$

$$a^{*} = \arg\max_{a} Q^{\pi}(s,a) \tag{3}$$

This action-value function is optimized by the interaction between the agent and environment with some RL algorithms, e.g., TD-learning, Q-learning, SARSA-learning (S.Sutton et al., 1998).

2.3 Model-based RL with trajectory sampling

To get the optimized value function efficiently, an agent can use an inner model with the Model-based RL algorithm. In the Model-based RL algorithm, it is important for the agent to generate some experiences using the inner model. These experiences using the inner model are called virtual experiences in this paper. One of the methods of generating the virtual experiences is the trajectory sampling method. In trajectory sampling, the virtual experiences are generated from the current state. In Model-based RL with trajectory sampling, at first, the agent chooses a next action virtually with the policy, e.g., $\varepsilon - greedy$, and it predicts a next state and a next reward from the current state and the chosen action using the inner model. Then, the agent updates the value function. Finally the agent sets the current state to the predicted state and sets current reward to predicted reward. The agent gets some experience by the continuation of this process. Fig.3 shows the concrete Model-

based RL algorithm with trajectory sampling. In this algorithm $policy(s,Q)$ is the action on s and Q with some policy, $Model(s,a)$ is the inner model, N_p is the number of indirectory learning, N_l is the length of trajectory, $\alpha(0 \le \alpha \le 1)$ is a step size parameter, and $\gamma(0 \le \gamma \le 1)$ is a discount rate.

(0) Initialize all variables and $a \leftarrow policy(s,Q)$
Do forever:
(1) $s \leftarrow$ current state
(2) Execute action a, Observe resultant state s' and reward r
(3) $a' \leftarrow policy(s,Q)$
(4) $Q(s,a) \leftarrow Q(s,a) + \alpha[\hat{r} + \gamma Q(s',a') - Q(s,a)]$
(5) Update $Model(s,a)$
(6) Repeat N_p times:
(6-1) $\hat{s} \leftarrow s$, $\hat{a} \leftarrow a$
(6-2)Repeat N_l times:
(6-2-1) s', $r \leftarrow Model(s,a)$、$\hat{s}' \leftarrow s'$, $\hat{r} \leftarrow r$
(6-2-2) $\hat{a}' \leftarrow policy(s,Q)$
(6-2-3) Push $(\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}')$ on stack
(6-2-4) $\hat{s} \leftarrow \hat{s}'$、$\hat{a} \leftarrow \hat{a}'$
(6-3) while stack has some item:
(6-3-1) Pop $(\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}')$
(6-3-2) $Q(\hat{s},\hat{a}) \leftarrow Q(\hat{s},\hat{a}) + \alpha[\hat{r} + \gamma Q(\hat{s}',\hat{a}') - Q(\hat{s},\hat{a})]$
(7) $s \leftarrow s'$、$a \leftarrow a'$
($\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}'$ are temporary variable)

Fig. 3. Algorithm of Model-based RL with Trajectory Sampling

3. Model error and MEFP-RL

This section describes the effect of some model error on Model-based RL, and introduces the ME-FRPL algorithm that can reduce its effect.

3.1 Effect of model error

An agent of Model-based RL can learn behavior efficiently when virtual experiences are equal to real experiences that are generated by environment. On the other hands, some virtual experiences different from real experiences give some bad effects to learning. These effects depend on the amount of the model error. Then, let us discuss the effect of the model on Model-based RL with trajectory sampling, where it is assumed that $N_p = 1$, $N_l = 1$.

The learning rate multiplied by TD-Error (S.Sutton et al., 1998) usually denotes the amount of learning with one update in RL. Then, the amount of learning with real experiences and that with virtual experiences are denoted as the amount of direct learning and as the

amount of indirect learning, respectively in this paper. The amount of direct learning Δ_d is given by Eq. (4) and the amount of indirect learning Δ_i is given by Eq. (5), where α_d is a direct learning rate, α_i is an indirect learnign rate, and the sign of $\wedge$ shows the prediction by the inner model. Eq. (5) is rewritten to Eq. (6) because frist $\hat{s}$ is s and first $\hat{a}$ is a in Q (See (6-1) in Fig.3), and when $N_p = 1$ and $N_l = 1$. So, the amount of all learning $\Delta_d + \Delta_i$ is given by Eq.(7).

$$\Delta_d = \alpha_d[r + \gamma Q(s', a') - Q(s, a)] \tag{4}$$

$$\Delta_i = \alpha_i[r + \gamma Q(\hat{s}', \hat{a}') - Q(\hat{s}, \hat{a})] \tag{5}$$

$$\Delta_i = \alpha_i[r + \gamma Q(\hat{s}', \hat{a}') - Q(s, a)] \tag{6}$$

$$\Delta_d + \Delta_i = \alpha_d[r + \gamma Q(s', a') - Q(s, a)] + \alpha_i[r + \gamma Q(\hat{s}', \hat{a}') - Q(s, a)] \tag{7}$$

Here, let the state dissimilarity $sds(s_a, s_b)$ between state s_a and state s_b be defined as the number of the necessary action for giving $s_a = s_b$. In this argument, if the agent can change s_a to s_b by K steps of actions, it is assumed that the agent can also change s_b to s_a with the same number (K step) of actions, though the former actions are usually different from the latter ones. Q is multiplied by γ each getting away from the goal when the agent does not gets some reward. If learning is almost convergent and the agent chooses only greedy action, $sds(s', \hat{s}')$ is constant. When s' is closer to a goal than $\hat{s}'$, then $Q(\hat{s}', \hat{a}') \geq Q(s', a')$. When $\hat{s}'$ is closer to a goal than s', then $Q(\hat{s}', \hat{a}') \leq Q(s', a')$. That is, $Q(\hat{s}', \hat{a}')$ is smaller than $\gamma^{-sds(s', \hat{s}')} Q(s', a')$ and $Q(\hat{s}', \hat{a}')$ is larger than $\gamma^{sds(s', \hat{s}')} Q(s', a')$. The relation between $Q(s', a')$ and $Q(\hat{s}', \hat{a}')$ is given by Eq. (8) using equilibrium parameter k.

$$Q(\hat{s}', \hat{a}') = \gamma^k Q(s', a') \quad (-sds(s', \hat{s}') \leq k \leq sds(s', \hat{s}')) \tag{8}$$

Now, applying Eq.(8) to Eq.(7), the amount of all learning $\Delta_d + \Delta_i$ is rewritten to Eq.(9).

$$\Delta_d + \Delta_i = (\alpha_d + \alpha_i) r - (\alpha_d + \alpha_i) Q(s, a) + (\alpha_d \gamma + \alpha_i \gamma^{k+1}) Q(s', a') \tag{9}$$

When k becomes very small minus value, $\alpha_i \gamma^{k+1}$ becomes to very big value. If the learning is convergence , $\Delta_d > 0$, and $Q(s, a)$ is only effected by $Q(s', a')$, $\gamma Q(s', a') = Q(s, a)$. So $\Delta_d + \Delta_i$ is given by (10).

$$\Delta_d + \Delta_i = (\alpha_d + \alpha_i) r + \alpha_i Q(s, a)(\gamma^k - 1) \tag{10}$$

If the model does not have any error, $\Delta_d + \Delta_i = (\alpha_d + \alpha_i) r$. That is, some inner model error makes some effect exponentially with change of k.

3.2 Decreasing effect of model error

Let α_i be defined by $\alpha_d h\gamma^{|k|}$ $(0 \le h \le 1)$. Then, $\Delta_d + \Delta_i$ is given by (11). Let us discuss the effect of some inner model error.

$$\begin{aligned}\Delta_d + \Delta_i &= (\alpha_d + \alpha_d h\gamma^{|k|})r - (\alpha_d + \alpha_d h\gamma^{|k|})Q(s,a) + (\alpha_d\gamma + \alpha_d h\gamma^{|k|}\gamma^{k+1})Q(s',a') \\ &= \alpha_d\{(1 + h\gamma^{|k|})r - (1 + h\gamma^{|k|})Q(s,a) + (\gamma + h\gamma^{|k|}\gamma^{k+1})Q(s',a')\}\end{aligned} \tag{11}$$

1. When $k \ge 0$, $\Delta_d + \Delta_i$ is rewritten to Eq.(12).

$$\Delta_d + \Delta_i = \alpha_d\{(1 + h\gamma^{k})r - (1 + h\gamma^{k})Q(s,a) + (\gamma + h\gamma^{k}\gamma^{k+1})Q(s',a')\} \tag{12}$$

   In this case, if k becomes large, $\Delta_d + \Delta_i$ becomes Δ_d. That is, the effect of the learning is reduced.
2. When $k < 0$, $\Delta_d + \Delta_i$ is rewritten to Eq.(13).

$$\Delta_d + \Delta_i = \alpha_d\{(1 + h\gamma^{-k})r - (1 + h\gamma^{-k})Q(s,a) + (\gamma + h\gamma)Q(s',a')\} \tag{13}$$

   In this case, if k becomes small, $\Delta_d + \Delta_i$ becomes $\Delta_d + \alpha_d h\gamma Q(s',a')$. That is, the effect of the learning is $\alpha_d h\gamma Q(s',a')$.

From these results, the model error does not make effect exponentially when α_i is equal to $\alpha_d h\gamma^{|k|}$ because the effect of the learning is smaller than $\alpha_d h\gamma Q(s',a')$. However, k is an unknown value because the relation between $Q(s',a')$ and $Q(\hat{s}',\hat{a}')$ is changed every learning procedure and every update. On the other hand, $sds(s',\hat{s}')$ is estimated by the model error. Because the relation beteween k and $sds(s',\hat{s}')$ is given by Eq.(8)($|k| \le sds(s',\hat{s}')$), Eq.(14) is obatined.

$$h\gamma^{sds(s',\hat{s}')} \le h\gamma^{|k|} \tag{14}$$

Threfore, the model error does not make effect exponentially when α_i is $\alpha_d h\gamma^{sds(s',\hat{s}')}$ instead of $\alpha_d h\gamma^{|k|}$.

3.3 ME-FPRL

On the basis of these results discussed in 3.2, Model Error based Forward Planning Reinforcement Learning (ME-FPRL) algorithm is propsed. Fig.4 shows the algorithm of ME-FPRL, where $\alpha(0 \le \alpha \le 1)$ is a step size parameter, $\gamma(0 \le \gamma \le 1)$ is a discount rate and ρ is a tradeoff parameter. This algorithm is based on Model-Based RL with the Trajectory Sampling and controls the amount of indirect learning by estimating the current error. Therefore, the proposed learning algorithm has robustness against the error. *ModelAcy* is estimated at state dissimilarity that is obtained by the current state dissimilarity, where the current state dissimilarity is defined as the latest number of necessary actions. In this algorithm, when ρ is nearly equal to 1, the latest state dissimilarity $sds(s',\hat{s}')$ affects *ModelAcy* very well, and on the other hand when ρ is nearly equal to 0, the previous state

dissimilarity affects $ModelAcy$ very well. N_l, N_p and N_{plan} are parameter values of indirect learning. N_l is the length of the trajectory, N_p is the number of indirect learning, and h is the parameter to change ratio of indirect learning.

(0) Initialize all variable and $a \leftarrow policy(s,Q)$
Do forever:
(1) $s \leftarrow$ current state
(2) Execute action a, Observe resultant state s' and reward r
(3) $a' \leftarrow policy(s,Q)$
(4) $Q(s,a) \leftarrow Q(s,a) + \alpha[\hat{r} + \gamma Q(s',a') - Q(s,a)]$
(5) Update $Model(s,a)$
(6) $ModelAcy(s,a) \leftarrow (1-\rho) \cdot ModelAcy(s,a) + \rho \cdot \gamma^{sds(s',\hat{s}')}$
(7) Repeat N_p times
(7-1) $\hat{s} \leftarrow s$、$\hat{a} \leftarrow a$、$ComAcy \leftarrow 1$
(7-2)Repeat N_l times:
(7-2-1) s'、$r \leftarrow Model(s,a)$、$\hat{s}' \leftarrow s'$、$\hat{r} \leftarrow r$
(7-2-2) $\hat{a}' \leftarrow policy(s,Q)$
(7-2-3) $ComAcy \leftarrow ComAcy \cdot ModelAcy(\hat{s},\hat{a})$
(7-2-3) Push $(\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}',ComAcy)$ on stack
(7-2-4) $\hat{s} \leftarrow \hat{s}'$、$\hat{a} \leftarrow \hat{a}'$
(7-3) while stack has some item:
(7-3-1) Pop $(\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}',ComAcy)$
(7-3-2) $Q(\hat{s},\hat{a}) \leftarrow Q(\hat{s},\hat{a}) + ComAcy \cdot \alpha h / N_{plan}[\hat{r} + \gamma Q(\hat{s}',\hat{a}') - Q(\hat{s},\hat{a})]$
(8) $s \leftarrow s'$、$a \leftarrow a'$
($\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}',ComAcy$ are temporary variable)

Fig. 4. Algorithm of ME-FPRL

4. Application to pursuing target task

In this section, at first, ME-FPRL is extended to a liner method for the continuous state space. Then, ME-FPRL is applied to pursuing target task and the efficiency of ME-FPRL is shown.

4.1 Continuous state spaces

In a real world, some tasks such as the control of robots are very complex because the state space is continuous and very large. Therefore, an agent needs much time to learn some behaviors on the continuous state space. This problem is known as *curse of dimensionality* (S.Sutton et al., 1998). To deal with this problem, there is a method to convert the original state space into the feature space with enough size to be able to solve the problem. This converting method is called *function approximation* (S.Sutton et al., 1998). One of function

approximation methods is the liner method. In the liner method, the action-value function Q is defined by Eq. (15).

$$Q(s,a) = \sum_{i=1}^{I} \theta(i,a)\phi(i,s) , \tag{15}$$

where s is a state, $\theta(i,a)$ is a parameter, $\phi(i,s)$ $(0 \le \phi(i,s) \le 1)$ is a feature vector, i is an index of a feature vector and I is a size of feature vector. $Q(s,a)$ is defined as summation of products of $\theta(i,a)$ and $\phi(i,s)$. That is, $Q(s,a)$ is a linear function of $\phi(i,s)$. The action-value function is optimized by the general gradient-descent update, and if a feature vector is given by the linear approximation method, the action-value function converges to a local optimum (S.Sutton et al., 1998). ME-FPRL is modified by the liner method in order to apply ME-FPRL to some practical tasks. Fig.5 shows the algorithm of ME-FPRL with the function approximation.

4.2 Pursuing target task

(0) Initialize all variable and $a \leftarrow policy(s,Q)$
Do forever:
(1) $s \leftarrow$ current state
(2) Execute action a, Observe resultant state s' and reward r
(3) $a' \leftarrow policy(s,Q)$
(4) $\delta \leftarrow r + \gamma \sum_{i=1}^{n} \theta(i,a')\phi(i,s') - \sum_{i=1}^{n} \theta(i,a)\phi(i,s)$
foreach i in I : $\theta(i,a) \leftarrow \theta(i,a) + \alpha \cdot \phi(i,s) \cdot \delta$
(5) Update $Model(s,a)$
(6) $ModelAcy(s,a) \leftarrow (1-\rho) \cdot ModelAcy(s,a) + \rho \cdot \gamma^{sds(s',\hat{s}')}$
(7) Repeat N_p times
(7-1) $\hat{s} \leftarrow s$ 、 $\hat{a} \leftarrow a$ 、 $ComAcy \leftarrow 1$
(7-2)Repeat N_l times:
(7-2-1) s' 、 $r \leftarrow Model(s,a)$ 、 $\hat{s}' \leftarrow s'$ 、 $\hat{r} \leftarrow r$
(7-2-2) $\hat{a}' \leftarrow policy(s,Q)$
(7-2-3) $ComAcy \leftarrow ComAcy \cdot ModelAcy(\hat{s},\hat{a})$
(7-2-3) Push $(\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}',ComAcy)$ on stack
(7-2-4) $\hat{s} \leftarrow \hat{s}'$ 、 $\hat{a} \leftarrow \hat{a}'$
(7-3) while stack has some item:
(7-3-1) Pop $(\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}',ComAcy)$
(7-3-2) $\delta \leftarrow \hat{r} + \gamma \sum_{i=1}^{n} \theta(i,\hat{a}')\phi(i,\hat{s}') - \sum_{i=1}^{n} \theta(i,\hat{a})\phi(i,\hat{s})$
foreach i in I : $\theta(\hat{s},\hat{a}) \leftarrow \theta(\hat{s},\hat{a}) + ComAcy \cdot \alpha h / N_{plan} \cdot \delta$
(8) $s \leftarrow s'$ 、 $a \leftarrow a'$
($\hat{s},\hat{a},\hat{r},\hat{s}',\hat{a}',ComAcy,\delta$ are temporary variable)

Fig.5. Algorithm of ME-FPRL with function approximation

The pursuing target task is a basic problem in the control of a robot system and many control methods are proposed for this problem. However, a control policy adjusted to a robot cannot be applied to another robot system because of different dynamics of each robot system. That is, adjustment of control policy that adapts dynamics is important for each robot system, and machine learning is one of the solutions to control it. In this study, ME-FPRL is applied to the pursuing target task. A four-legged robot shown in Fig.6 (SONY AIBO ERS-7M3) acquires the control policy to pursue a target shown in Fig.7 by learning. This robot has a CCD camera moving pan/tilt directions. Fig. 6 also shows the default position. The robot can recognize the target using simple image processing. And, the agent can sense the direction of the camera. The target is moved like a pendulum by the servomotor. In this section, this robot is called an agent simply.

4.3 Action and reward

In this task, the purpose of the agent is that it learns behavior that can continue catching the target in the center of the camera. In order to catch the target, the agent chooses one of five kinds of actions: *Turn to top*, *Turn to bottom*, *Turn to right*, *Turn to left*, and *Stop*. When the agent chooses *Turn to top* or *Turn to bottom*, the tilt direction of the camera is changed. When the agent chooses *Turn to right* or *Turn to left*, the pan direction of the camera is changed. These changes are for every five degrees. When the agent chooses *Stop*, the direction of the camera is not changed. The state space is expressed by the target position with axes of pan and tilt, and the agent changes its state by these 5 kinds of actions.

The agent gets one of the following three kinds of rewards: 10 points; catching a target at near the center of the camera, -10 points; catching a target out of the center of the camera, -20 points; missing a target. The agent gets best policy based on these rewards.

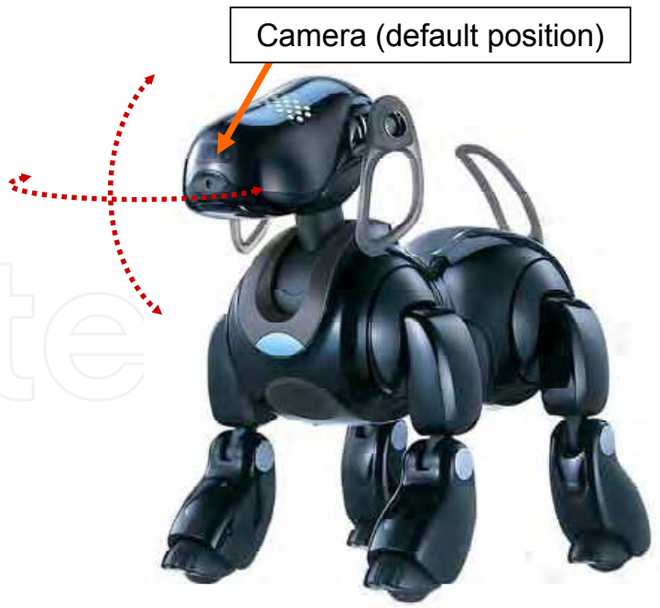


Fig. 6. A Four-legged Robot

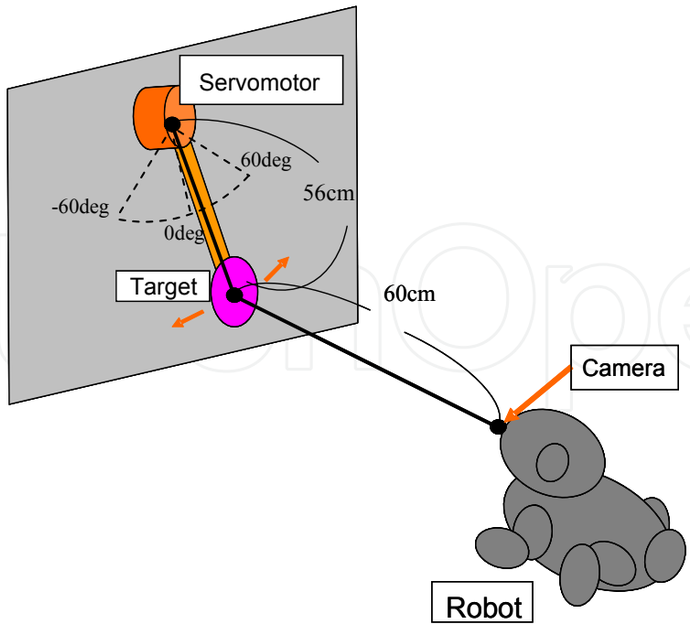


Fig. 7. Relation between robot and target

4.4 Learning agents

In order to confirm the validity of ME-FPRL, the learning efficiencies of the control policy with RL, model-based RL and ME-FPRL are compared with each other.

An agents' action is chosen by Gibbs (or Boltzmann) Sampler (S.Sutton et al., 1998) based on Eq.(16) showing Gibbs distribution, where τ is a time constant. The agent using Gibbus Sampler chooses one of the actions like $\varepsilon - greedy$ when τ is equal to 0, and chooses one of the actions with equal probability when τ is equal to 1.

$$Gibbs(s,Q,a) = \frac{\exp(Q(s,a)/\tau)}{\sum_{b \in A} \exp(Q(s,b)/\tau)} \tag{16}$$

The agent approximates state spaces by the CMAC (S.Sutton et al., 1998), that is a liner approximation method. In the CMAC, state spaces are expressed by some tiles. The number of tiles is defined as $Tiling$. In this application, parameter values are set as follows: $Tiling = 50$, $\alpha = 0.2/Tiling$, $\gamma = 0.8$, $\rho = 0.3$, $N_p = 8$, $N_l = 3$, $N_{plan} = N_p/3$, $h = 0.9$, $\tau = 0.4$. θ and the initial value of $ModelAcy$ are 0.

4.5 Inner model and its learning

The agent with the model-based method has an inner model. In this application, dynamics of the target are constructed by multi-layered artificial neural networks (Nakano, et al., 2006). Fig.8 shows the inner model of the target having two networks (Net1, Net2). These networks are switched over by the direction of target's movement. That is, Net1 deals with the prediction of a right-handed rotation and Net2 deals with the prediction of a left-handed

rotation. These networks are obtained by using the error back propagation methods for the learning (Nakano, et al, 2006). Each network is constructed by an input layer with 18 nodes, a hidden layer with 20 nodes and an output layer with 18 nodes. Although the state space for RL is expressed in the relative coordinate value about target position, the state space for this model is expressed in absolute coordinate value about target position without depending on the direction of the camera in this model. Therefore, all predictions by this model do not

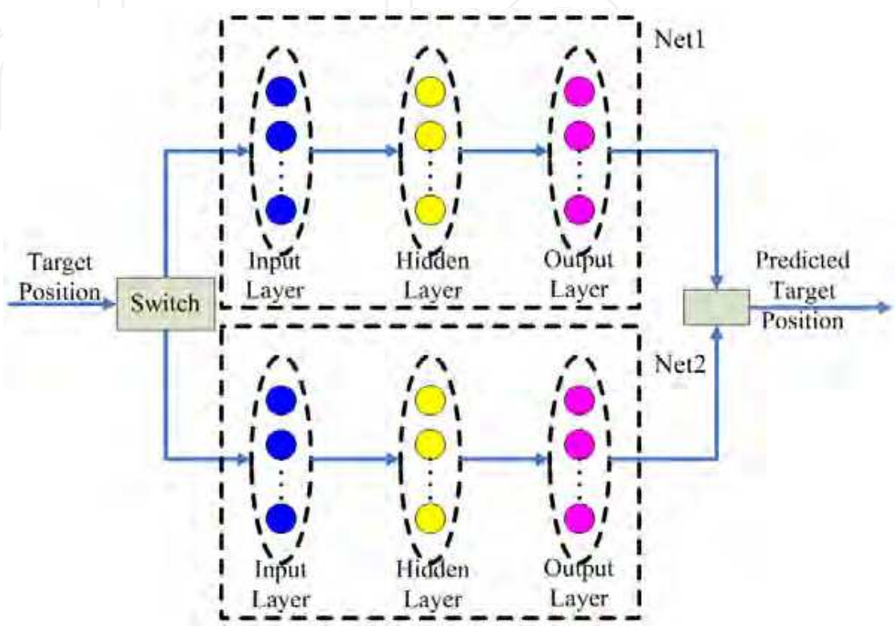


Fig. 8. Inner Model with Two Neural Networks

depend on camera direction. Before performing the task, the agent learns a trace of the target with this model. The target has a pendulum swing between 0 deg and 60 deg with constant speed. The agent memories 1000 times target positions that change momentarily. After the observation, the agent learns the trace by the memorized target positions.

4.6 Experiment and its result

In this experiment, the task is composed of 100 episodes, one episode ends when the robot catches a target in the center of a camera for 40 steps or when the robot loses the target completely. In the first half, 50 episodes, the target has a pendulum swing between 0 deg and 60 deg. In the latter half, 50 episodes, the target has a pendulum swing between 60 deg and -60 deg. That is, the trace of the target is changed at the 50th episodes. Therefore, the agent learns behavior against the unknown trace of a target. Fig.9 shows experimental results with RL, Fig.10 shows those with ME-FPRL and Fig.11 shows those with model-based RL. In these figures, a horizontal axis means the number of episodes, and a vertical axis means the number of steps that the agent catches the target. Table 1 shows the number of episode times that the agent catches the 40 steps target. The agent with Model-based RL catches the target st the 2nd episode, the most quickly among three three methods. The agent with ME-FPRL catches the target at the 8th episode, the second best. However, the agent with Model-based RL often misses the target even in the latter half. On the other hand

the agent with ME-FPRL can continue catching the target competely after the 8th episode. Althoufh the number of episode times that the agent with RL misses the target in the latter half is small comparing with the agent with Model-based RL, it takes the longest times to catches the target for the first time. Therefore, it is found that ME-FPRL is the most efficient in the early stage and the most robust aginst the disturbance among the three methods.

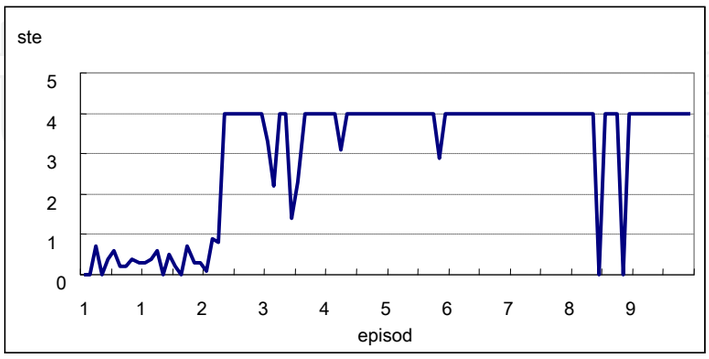


Fig. 9. Experimental Results of RL

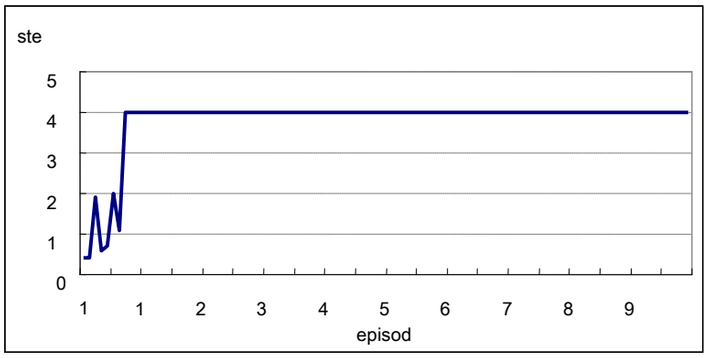


Fig. 10. Experimental Results of ME-FPRL

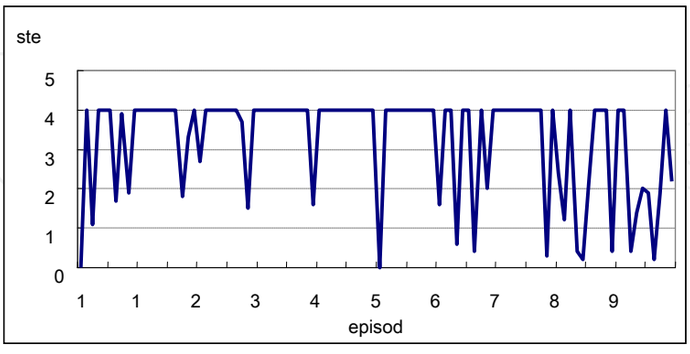


Fig. 11. Experimental Results of Model-based RL

Learning Methods	1st-50th episode	51st - 100th episode	Total (1st – 100th episode)
RL	22	47	69
ME-FPRL	43	50	93
Model-based RL	39	31	70

Table 1. The number of Episode Times that Agent Catches 40 Steps Target

5. Conclusions

In this chapter, the learning algorithm ME-FPRL is discussed. And it applied to the pursuit target task. Application results show that the ME-FPRL is more efficient than a RL or Model-based RL. As a result, ME-FPRL is found to be able to apply to practical tasks. Our future work is constructing more an efficient learning system by using some advice and communication.

6. References

Takakatsu Isozumi:Development of Humanoid Robot,Kawata Giho,Vol. 22, 2003

Shintaro Anzui, Satoshi Hukuda, Masahiro Hamasaki, Ikki Ohmukai, Hideaki Takeda, and Takahira Yamaguti. (2005). A Recommendation System for Personal Digital Assistance with Ontologies, *The 19th Annual Conference of the Japanese Society for Artificial Intelligence*

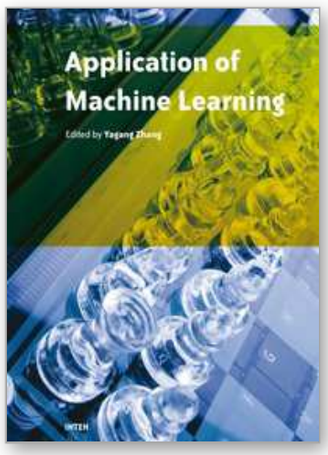
Richard S.Sutton and Andrew G.Garto (1998). *Reinforcement learning: an introduction*. The MIT Press

M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda (1994). Vision-Based Behavior Acquisition for a Shooting Robot by Using A Reinforcement Learning. *Proc. of IAPR/IEEE Workshop on Visual Behaviors-1994,* pp.112-118

Yoshiyuki Tajima, Takehisa Onisawa (2006). Model-based Reinforcement Learning with Model Error. *International Symposium on Computational Intelligence and Industrial Applications(ISCIIA)*, pp127-134, Guangzhou, China

David Muse, Kevin Burn, Stefan Wermter (2006). Reinforcement Learning for Platform-Independent Visual Robot Control. *IEEE World Congress on Computational Intelligence*,pp.4766-4773

Ryohey Nakano (2005). *Basic Theory of Neural Network Information Processing*. Surikogakusya



Application of Machine Learning
Edited by Yagang Zhang

ISBN 978-953-307-035-3
Hard cover, 280 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

The goal of this book is to present the latest applications of machine learning, which mainly include: speech recognition, traffic and fault classification, surface quality prediction in laser machining, network security and bioinformatics, enterprise credit risk evaluation, and so on. This book will be of interest to industrial engineers and scientists as well as academics who wish to pursue machine learning. The book is intended for both graduate and postgraduate students in fields such as computer science, cybernetics, system sciences, engineering, statistics, and social sciences, and as a reference for software professionals and practitioners. The wide scope of the book provides them with a good introduction to many application researches of machine learning, and it is also the source of useful bibliographical information.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen